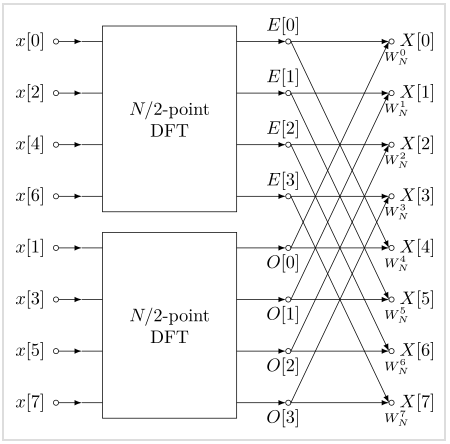
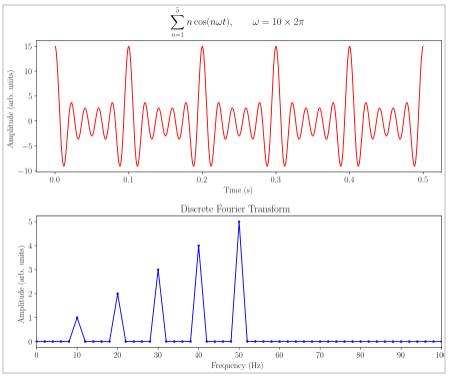
WIKIPEDIA
The Free Encyclopedia

Fast Fourier transform

A **fast Fourier transform** (**FFT**) is an algorithm that computes the discrete Fourier transform (DFT) of a sequence, or its inverse (IDFT). Fourier analysis converts a signal from its original domain (often time or space) to a representation in the frequency domain and vice versa. The DFT is obtained by decomposing a sequence of values into components of different frequencies.[1] This operation is useful in many fields, but computing it directly from the definition is often too slow to be practical. An FFT rapidly computes such transformations by factorizing the DFT matrix into a product of sparse (mostly zero) factors.[2] As a result, it manages to reduce the complexity of computing the DFT from $O(n^2)$, which arises if one simply applies the definition of DFT, to $O(n \log n)$, where n is the data size. The difference in speed can be enormous, especially for long data sets where n may be in the thousands or millions. In the presence of round-off error, many FFT algorithms are much more accurate than evaluating the DFT definition directly or indirectly. There are many different FFT algorithms based on a wide range of published theories, from simple complex-number arithmetic to group theory and number theory.



An example FFT algorithm structure, using a decomposition into half-size FFTs



A discrete Fourier analysis of a sum of cosine waves at 10, 20, 30, 40, and 50 Hz

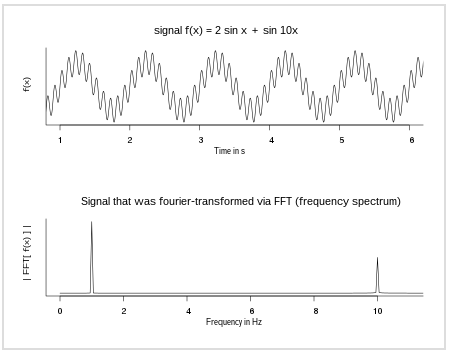
Fast Fourier transforms are widely used for applications in engineering, music, science, and mathematics. The basic ideas were popularized in 1965, but some algorithms had been derived as early as 1805.[1] In 1994, Gilbert Strang described the FFT as "the most important numerical algorithm of our lifetime",[3][4] and it was included in Top 10 Algorithms of 20th Century by the IEEE magazine *Computing in Science & Engineering*.[5]

The best-known FFT algorithms depend upon the factorization of n, but there are FFTs with $O(n \log n)$ complexity for all, even prime, n. Many FFT algorithms depend only on the fact that $e^{-2\pi i/n}$ is an n'th primitive root of unity, and thus can be applied to analogous transforms over any finite field, such as number-theoretic transforms. Since the inverse DFT is the same as the DFT, but with the opposite sign in the exponent and a $1/n$ factor, any FFT algorithm can easily be adapted for it.

History

The development of fast algorithms for DFT can be traced to Carl Friedrich Gauss's unpublished work in 1805 when he needed it to interpolate the orbit of asteroids Pallas and Juno from sample observations.[6][7] His method was very similar to the one published in 1965 by James Cooley and

John Tukey, who are generally credited for the invention of the modern generic FFT algorithm. While Gauss's work predated even Joseph Fourier's results in 1822, he did not analyze the computation time and eventually used other methods to achieve his goal.



Time-based representation (above) and frequency-based representation (below) of the same signal, where the lower representation can be obtained from the upper one by Fourier transformation

Between 1805 and 1965, some versions of FFT were published by other authors. Frank Yates in 1932 published his version called *interaction algorithm*, which provided efficient computation of Hadamard and Walsh transforms.[8] Yates' algorithm is still used in the field of statistical design and analysis of experiments. In 1942, G. C. Danielson and Cornelius Lanczos published their version to compute DFT for x-ray crystallography, a field where calculation of Fourier transforms presented a formidable bottleneck.[9][10] While many methods in the past had focused on reducing the constant factor for $O(n^2)$ computation by taking advantage of "symmetries", Danielson and Lanczos realized that one could use the "periodicity" and apply a "doubling trick" to "double [n] with only slightly more than double the labor", though like Gauss they did not analyze that this led to $O(n \log n)$ scaling.[11]

James Cooley and John Tukey independently rediscovered these earlier algorithms[7] and published a more general FFT in 1965 that is applicable when n is composite and not necessarily a power of 2, as well as analyzing the $O(n \log n)$ scaling.[12] Tukey came up with the idea during a meeting of President Kennedy's Science Advisory Committee where a discussion topic involved detecting nuclear tests by the Soviet Union by setting up sensors to surround the country from outside. To analyze the output of these sensors, an FFT algorithm would be needed. In discussion with Tukey, Richard Garwin recognized the general applicability of the algorithm not just to national security problems, but also to a wide range of problems including one of immediate interest to him, determining the periodicities of the spin orientations in a 3-D crystal of Helium-3.[13] Garwin gave Tukey's idea to Cooley (both worked at IBM's Watson labs) for implementation.[14] Cooley and Tukey published the paper in a relatively short time of six months.[15] As Tukey did not work at IBM, the patentability of the idea was doubted and the algorithm went into the public domain, which, through the computing revolution of the next decade, made FFT one of the indispensable algorithms in digital signal processing.

Definition

Let $x_0, \ldots, x_{n-1}$ be complex numbers. The DFT is defined by the formula

$$X_k = \sum_{m=0}^{n-1} x_m e^{-i 2\pi k m / n} \qquad k = 0, \ldots, n-1,$$

where $e^{i2\pi/n}$ is a primitive n'th root of 1.

Evaluating this definition directly requires $O(n^2)$ operations: there are n outputs X_k, and each output requires a sum of n terms. An FFT is any method to compute the same results in $O(n \log n)$ operations. All known FFT algorithms require $O(n \log n)$ operations, although there is no known proof that lower complexity is impossible.[16]

To illustrate the savings of an FFT, consider the count of complex multiplications and additions for $n = 4096$ data points. Evaluating the DFT's sums directly involves n^2 complex multiplications and $n(n-1)$ complex additions, of which $O(n)$ operations can be saved by eliminating trivial operations such as multiplications by 1, leaving about 30 million operations. In contrast, the radix-2 Cooley–Tukey algorithm, for n a power of 2, can compute the same result with only $(n/2)\log_2(n)$ complex multiplications (again, ignoring simplifications of multiplications by 1 and similar) and $n\log_2(n)$ complex additions, in total about 30,000 operations — a thousand times less than with direct evaluation. In practice, actual performance on modern computers is usually dominated by factors other than the speed of arithmetic operations and the analysis is a complicated subject (for example, see Frigo & Johnson, 2005),[17] but the overall improvement from $O(n^2)$ to $O(n\log n)$ remains.

Algorithms

Cooley–Tukey algorithm

By far the most commonly used FFT is the Cooley–Tukey algorithm. This is a divide-and-conquer algorithm that recursively breaks down a DFT of any composite size $n = n_1 n_2$ into many smaller DFTs of sizes n_1 and n_2, along with $O(n)$ multiplications by complex roots of unity traditionally called twiddle factors (after Gentleman and Sande, 1966).[18]

This method (and the general idea of an FFT) was popularized by a publication of Cooley and Tukey in 1965,[12] but it was later discovered[1] that those two authors had independently re-invented an algorithm known to Carl Friedrich Gauss around 1805[19] (and subsequently rediscovered several times in limited forms).

The best known use of the Cooley–Tukey algorithm is to divide the transform into two pieces of size n/2 at each step, and is therefore limited to power-of-two sizes, but any factorization can be used in general (as was known to both Gauss and Cooley/Tukey[1]). These are called the *radix-2* and *mixed-radix* cases, respectively (and other variants such as the split-radix FFT have their own names as well). Although the basic idea is recursive, most traditional implementations rearrange the algorithm to avoid explicit recursion. Also, because the Cooley–Tukey algorithm breaks the DFT into smaller DFTs, it can be combined arbitrarily with any other algorithm for the DFT, such as those described below.

Other FFT algorithms

There are FFT algorithms other than Cooley–Tukey.

For $n = n_1 n_2$ with coprime n_1 and n_2, one can use the prime-factor (Good–Thomas) algorithm (PFA), based on the Chinese remainder theorem, to factorize the DFT similarly to Cooley–Tukey but without the twiddle factors. The Rader–Brenner algorithm (1976)[20] is a Cooley–Tukey-like factorization but with purely imaginary twiddle factors, reducing multiplications at the cost of increased additions and reduced numerical stability; it was later superseded by the split-radix variant of Cooley–Tukey (which achieves the same multiplication count but with fewer additions and without sacrificing accuracy). Algorithms that recursively factorize the DFT into smaller operations other than DFTs include the Bruun and QFT algorithms. (The Rader–Brenner[20] and QFT algorithms were proposed for power-of-two sizes, but it is possible that they could be adapted to general composite n.

Bruun's algorithm applies to arbitrary even composite sizes.) Bruun's algorithm, in particular, is based on interpreting the FFT as a recursive factorization of the polynomial $z^n - 1$, here into real-coefficient polynomials of the form $z^m - 1$ and $z^{2m} + az^m + 1$.

Another polynomial viewpoint is exploited by the Winograd FFT algorithm,[21][22] which factorizes $z^n - 1$ into cyclotomic polynomials—these often have coefficients of 1, 0, or −1, and therefore require few (if any) multiplications, so Winograd can be used to obtain minimal-multiplication FFTs and is often used to find efficient algorithms for small factors. Indeed, Winograd showed that the DFT can be computed with only $O(n)$ irrational multiplications, leading to a proven achievable lower bound on the number of multiplications for power-of-two sizes; unfortunately, this comes at the cost of many more additions, a tradeoff no longer favorable on modern processors with hardware multipliers. In particular, Winograd also makes use of the PFA as well as an algorithm by Rader for FFTs of *prime* sizes.

Rader's algorithm, exploiting the existence of a generator for the multiplicative group modulo prime n, expresses a DFT of prime size n as a cyclic convolution of (composite) size $n - 1$, which can then be computed by a pair of ordinary FFTs via the convolution theorem (although Winograd uses other convolution methods). Another prime-size FFT is due to L. I. Bluestein, and is sometimes called the chirp-z algorithm; it also re-expresses a DFT as a convolution, but this time of the *same* size (which can be zero-padded to a power of two and evaluated by radix-2 Cooley–Tukey FFTs, for example), via the identity

$$nk = -\frac{(k-n)^2}{2} + \frac{n^2}{2} + \frac{k^2}{2}.$$

Hexagonal fast Fourier transform (HFFT) aims at computing an efficient FFT for the hexagonally-sampled data by using a new addressing scheme for hexagonal grids, called Array Set Addressing (ASA).

FFT algorithms specialized for real or symmetric data

In many applications, the input data for the DFT are purely real, in which case the outputs satisfy the symmetry

$$X_{n-k} = X_k^*$$

and efficient FFT algorithms have been designed for this situation (see e.g. Sorensen, 1987).[23][24] One approach consists of taking an ordinary algorithm (e.g. Cooley–Tukey) and removing the redundant parts of the computation, saving roughly a factor of two in time and memory. Alternatively, it is possible to express an *even*-length real-input DFT as a complex DFT of half the length (whose real and imaginary parts are the even/odd elements of the original real data), followed by $O(n)$ post-processing operations.

It was once believed that real-input DFTs could be more efficiently computed by means of the discrete Hartley transform (DHT), but it was subsequently argued that a specialized real-input DFT algorithm (FFT) can typically be found that requires fewer operations than the corresponding DHT algorithm (FHT) for the same number of inputs.[23] Bruun's algorithm (above) is another method that was initially proposed to take advantage of real inputs, but it has not proved popular.

There are further FFT specializations for the cases of real data that have even/odd symmetry, in which case one can gain another factor of roughly two in time and memory and the DFT becomes the discrete cosine/sine transform(s) (DCT/DST). Instead of directly modifying an FFT algorithm for these cases, DCTs/DSTs can also be computed via FFTs of real data combined with $O(n)$ pre- and post-processing.

Computational issues

Bounds on complexity and operation counts

Unsolved problem in computer science:

What is the lower bound on the complexity of fast Fourier transform algorithms? Can they be faster than $O(N \log N)$*?*

(more unsolved problems in computer science)

A fundamental question of longstanding theoretical interest is to prove lower bounds on the complexity and exact operation counts of fast Fourier transforms, and many open problems remain. It is not rigorously proved whether DFTs truly require $\Omega(n \log n)$ (i.e., order $n \log n$ or greater) operations, even for the simple case of power of two sizes, although no algorithms with lower complexity are known. In particular, the count of arithmetic operations is usually the focus of such questions, although actual performance on modern-day computers is determined by many other factors such as cache or CPU pipeline optimization.

Following work by Shmuel Winograd (1978),[21] a tight $\Theta(n)$ lower bound is known for the number of real multiplications required by an FFT. It can be shown that only $4n - 2\log_2^2(n) - 2\log_2(n) - 4$ irrational real multiplications are required to compute a DFT of power-of-two length $n = 2^m$. Moreover, explicit algorithms that achieve this count are known (Heideman & Burrus, 1986;[25] Duhamel, 1990[26]). However, these algorithms require too many additions to be practical, at least on modern computers with hardware multipliers (Duhamel, 1990;[26] Frigo & Johnson, 2005).[17]

A tight lower bound is not known on the number of required additions, although lower bounds have been proved under some restrictive assumptions on the algorithms. In 1973, Morgenstern[27] proved an $\Omega(n \log n)$ lower bound on the addition count for algorithms where the multiplicative constants have bounded magnitudes (which is true for most but not all FFT algorithms). Pan (1986)[28] proved an $\Omega(n \log n)$ lower bound assuming a bound on a measure of the FFT algorithm's "asynchronicity", but the generality of this assumption is unclear. For the case of power-of-two n, Papadimitriou (1979)[29] argued that the number $n \log_2 n$ of complex-number additions achieved by Cooley–Tukey algorithms is *optimal* under certain assumptions on the graph of the algorithm (his assumptions imply, among other things, that no additive identities in the roots of unity are exploited). (This argument would imply that at least $2N \log_2 N$ real additions are required, although this is not a tight bound because extra additions are required as part of complex-number multiplications.) Thus far, no published FFT algorithm has achieved fewer than $n \log_2 n$ complex-number additions (or their equivalent) for power-of-two n.

A third problem is to minimize the *total* number of real multiplications and additions, sometimes called the "arithmetic complexity" (although in this context it is the exact count and not the asymptotic complexity that is being considered). Again, no tight lower bound has been proven. Since 1968, however, the lowest published count for power-of-two n was long achieved by the split-radix FFT algorithm, which requires $4n \log_2(n) - 6n + 8$ real multiplications and additions for $n > 1$. This

was recently reduced to $\sim \frac{34}{9} n \log_2 n$ (Johnson and Frigo, 2007;[16] Lundy and Van Buskirk, 2007[30]). A slightly larger count (but still better than split radix for $n \geq 256$) was shown to be provably optimal for $n \leq 512$ under additional restrictions on the possible algorithms (split-radix-like flowgraphs with unit-modulus multiplicative factors), by reduction to a satisfiability modulo theories problem solvable by brute force (Haynal & Haynal, 2011).[31]

Most of the attempts to lower or prove the complexity of FFT algorithms have focused on the ordinary complex-data case, because it is the simplest. However, complex-data FFTs are so closely related to algorithms for related problems such as real-data FFTs, discrete cosine transforms, discrete Hartley transforms, and so on, that any improvement in one of these would immediately lead to improvements in the others (Duhamel & Vetterli, 1990).[32]

Approximations

All of the FFT algorithms discussed above compute the DFT exactly (i.e. neglecting floating-point errors). A few "FFT" algorithms have been proposed, however, that compute the DFT *approximately*, with an error that can be made arbitrarily small at the expense of increased computations. Such algorithms trade the approximation error for increased speed or other properties. For example, an approximate FFT algorithm by Edelman et al. (1999)[33] achieves lower communication requirements for parallel computing with the help of a fast multipole method. A wavelet-based approximate FFT by Guo and Burrus (1996)[34] takes sparse inputs/outputs (time/frequency localization) into account more efficiently than is possible with an exact FFT. Another algorithm for approximate computation of a subset of the DFT outputs is due to Shentov et al. (1995).[35] The Edelman algorithm works equally well for sparse and non-sparse data, since it is based on the compressibility (rank deficiency) of the Fourier matrix itself rather than the compressibility (sparsity) of the data. Conversely, if the data are sparse—that is, if only k out of n Fourier coefficients are nonzero—then the complexity can be reduced to $O(k \log n \log n/k)$, and this has been demonstrated to lead to practical speedups compared to an ordinary FFT for $n/k > 32$ in a large-n example ($n = 2^{22}$) using a probabilistic approximate algorithm (which estimates the largest k coefficients to several decimal places).[36]

Accuracy

FFT algorithms have errors when finite-precision floating-point arithmetic is used, but these errors are typically quite small; most FFT algorithms, e.g. Cooley–Tukey, have excellent numerical properties as a consequence of the pairwise summation structure of the algorithms. The upper bound on the relative error for the Cooley–Tukey algorithm is $O(\varepsilon \log n)$, compared to $O(\varepsilon n^{3/2})$ for the naïve DFT formula,[18] where ε is the machine floating-point relative precision. In fact, the root mean square (rms) errors are much better than these upper bounds, being only $O(\varepsilon \sqrt{\log n})$ for Cooley–Tukey and $O(\varepsilon \sqrt{n})$ for the naïve DFT (Schatzman, 1996).[37] These results, however, are very sensitive to the accuracy of the twiddle factors used in the FFT (i.e. the trigonometric function values), and it is not unusual for incautious FFT implementations to have much worse accuracy, e.g. if they use inaccurate trigonometric recurrence formulas. Some FFTs other than Cooley–Tukey, such as the Rader–Brenner algorithm, are intrinsically less stable.

In fixed-point arithmetic, the finite-precision errors accumulated by FFT algorithms are worse, with rms errors growing as $O(\sqrt{n})$ for the Cooley–Tukey algorithm (Welch, 1969).[38] Achieving this accuracy requires careful attention to scaling to minimize loss of precision, and fixed-point FFT

algorithms involve rescaling at each intermediate stage of decompositions like Cooley–Tukey.

To verify the correctness of an FFT implementation, rigorous guarantees can be obtained in $O(n \log n)$ time by a simple procedure checking the linearity, impulse-response, and time-shift properties of the transform on random inputs (Ergün, 1995).[39]

Multidimensional FFTs

As defined in the multidimensional DFT article, the multidimensional DFT

$$X_{\mathbf{k}} = \sum_{\mathbf{n}=0}^{\mathbf{N}-1} e^{-2\pi i \mathbf{k} \cdot (\mathbf{n}/\mathbf{N})} x_{\mathbf{n}}$$

transforms an array $x_{\mathbf{n}}$ with a d-dimensional vector of indices $\mathbf{n} = (n_1, \ldots, n_d)$ by a set of d nested summations (over $n_j = 0 \ldots N_j - 1$ for each j), where the division $\mathbf{n}/\mathbf{N} = (n_1/N_1, \ldots, n_d/N_d)$ is performed element-wise. Equivalently, it is the composition of a sequence of d sets of one-dimensional DFTs, performed along one dimension at a time (in any order).

This compositional viewpoint immediately provides the simplest and most common multidimensional DFT algorithm, known as the **row-column** algorithm (after the two-dimensional case, below). That is, one simply performs a sequence of d one-dimensional FFTs (by any of the above algorithms): first you transform along the n_1 dimension, then along the n_2 dimension, and so on (actually, any ordering works). This method is easily shown to have the usual $O(n \log n)$ complexity, where $n = n_1 \cdot n_2 \cdots n_d$ is the total number of data points transformed. In particular, there are n/n_1 transforms of size n_1, etc., so the complexity of the sequence of FFTs is:

$$\begin{aligned} &\frac{n}{n_1} O(n_1 \log n_1) + \cdots + \frac{n}{n_d} O(n_d \log n_d) \\ &= O\left(n \left[\log n_1 + \cdots + \log n_d\right]\right) = O(n \log n). \end{aligned}$$

In two dimensions, the $x_{\mathbf{k}}$ can be viewed as an $n_1 \times n_2$ matrix, and this algorithm corresponds to first performing the FFT of all the rows (resp. columns), grouping the resulting transformed rows (resp. columns) together as another $n_1 \times n_2$ matrix, and then performing the FFT on each of the columns (resp. rows) of this second matrix, and similarly grouping the results into the final result matrix.

In more than two dimensions, it is often advantageous for cache locality to group the dimensions recursively. For example, a three-dimensional FFT might first perform two-dimensional FFTs of each planar "slice" for each fixed n_1, and then perform the one-dimensional FFTs along the n_1 direction. More generally, an asymptotically optimal cache-oblivious algorithm consists of recursively dividing the dimensions into two groups $(n_1, \ldots, n_{d/2})$ and $(n_{d/2+1}, \ldots, n_d)$ that are transformed recursively (rounding if d is not even) (see Frigo and Johnson, 2005).[17] Still, this remains a straightforward variation of the row-column algorithm that ultimately requires only a one-dimensional FFT algorithm as the base case, and still has $O(n \log n)$ complexity. Yet another variation is to perform matrix transpositions in between transforming subsequent dimensions, so that the transforms operate on contiguous data; this is especially important for out-of-core and distributed memory situations where accessing non-contiguous data is extremely time-consuming.

There are other multidimensional FFT algorithms that are distinct from the row-column algorithm, although all of them have $O(n \log n)$ complexity. Perhaps the simplest non-row-column FFT is the vector-radix FFT algorithm, which is a generalization of the ordinary Cooley–Tukey algorithm where one divides the transform dimensions by a vector $\mathbf{r} = (r_1, r_2, \ldots, r_d)$ of radices at each step. (This may also have cache benefits.) The simplest case of vector-radix is where all of the radices are equal (e.g. vector-radix-2 divides *all* of the dimensions by two), but this is not necessary. Vector radix with only a single non-unit radix at a time, i.e. $\mathbf{r} = (1, \ldots, 1, r, 1, \ldots, 1)$, is essentially a row-column algorithm. Other, more complicated, methods include polynomial transform algorithms due to Nussbaumer (1977),[40] which view the transform in terms of convolutions and polynomial products. See Duhamel and Vetterli (1990)[32] for more information and references.

Other generalizations

An $O(n^{5/2} \log n)$ generalization to spherical harmonics on the sphere s^2 with n^2 nodes was described by Mohlenkamp,[41] along with an algorithm conjectured (but not proven) to have $O(n^2 \log^2(n))$ complexity; Mohlenkamp also provides an implementation in the libftsh library.[42] A spherical-harmonic algorithm with $O(n^2 \log n)$complexity is described by Rokhlin and Tygert.[43]

The fast folding algorithm is analogous to the FFT, except that it operates on a series of binned waveforms rather than a series of real or complex scalar values. Rotation (which in the FFT is multiplication by a complex phasor) is a circular shift of the component waveform.

Various groups have also published "FFT" algorithms for non-equispaced data, as reviewed in Potts *et al.* (2001).[44] Such algorithms do not strictly compute the DFT (which is only defined for equispaced data), but rather some approximation thereof (a non-uniform discrete Fourier transform, or NDFT, which itself is often computed only approximately). More generally there are various other methods of spectral estimation.

Applications

The FFT is used in digital recording, sampling, additive synthesis and pitch correction software.[45]

The FFT's importance derives from the fact that it has made working in the frequency domain equally computationally feasible as working in the temporal or spatial domain. Some of the important applications of the FFT include:[15][46]

- fast large-integer multiplication algorithms and polynomial multiplication,
- efficient matrix–vector multiplication for Toeplitz, circulant and other structured matrices,
- filtering algorithms (see overlap–add and overlap–save methods),
- fast algorithms for discrete cosine or sine transforms (e.g. fast DCT used for JPEG and MPEG/MP3 encoding and decoding),
- fast Chebyshev approximation,
- solving difference equations,
- computation of isotopic distributions.[47]
- modulation and demodulation of complex data symbols using orthogonal frequency division multiplexing (OFDM) for 5G, LTE, Wi-Fi, DSL, and other modern communication systems.

An original application of the FFT in finance particularly in the Valuation of options was developed by Marcello Minenna.[48]

Research areas

Big FFTs
With the explosion of big data in fields such as astronomy, the need for 512K FFTs has arisen for certain interferometry calculations. The data collected by projects such as WMAP and LIGO require FFTs of tens of billions of points. As this size does not fit into main memory, so called out-of-core FFTs are an active area of research.[49]

Approximate FFTs
For applications such as MRI, it is necessary to compute DFTs for nonuniformly spaced grid points and/or frequencies. Multipole based approaches can compute approximate quantities with factor of runtime increase.[50]

Group FFTs
The FFT may also be explained and interpreted using group representation theory allowing for further generalization. A function on any compact group, including non-cyclic, has an expansion in terms of a basis of irreducible matrix elements. It remains active area of research to find efficient algorithm for performing this change of basis. Applications including efficient spherical harmonic expansion, analyzing certain Markov processes, robotics etc.[51]

Quantum FFTs
Shor's fast algorithm for integer factorization on a quantum computer has a subroutine to compute DFT of a binary vector. This is implemented as sequence of 1- or 2-bit quantum gates now known as quantum FFT, which is effectively the Cooley–Tukey FFT realized as a particular factorization of the Fourier matrix. Extension to these ideas is currently being explored.[52]

Language reference

Language	`Command/Method`	Pre-requisites
R	`stats::fft(x)`	None
Scilab	`fft(x)`	None
Octave/MATLAB	`fft(x)`	None
Python	`fft.fft(x)`	numpy or scipy
Mathematica	`Fourier[x]`	None
Fortran	`fftw_one(plan,in,out)`	FFTW
Julia	`fft(A [,dims])`	FFTW
Rust	`fft.process(&mut x);`	rustfft (https://docs.rs/rustfft/latest/rustfft/)
Haskell	`dft x`	fft (https://hackage.haskell.org/package/fft)

See also

FFT-related algorithms:

- Bit-reversal permutation
- Goertzel algorithm – computes individual terms of discrete Fourier transform

FFT implementations:

- ALGLIB – a dual/GPL-licensed C++ and C# library (also supporting other languages), with real/complex FFT implementation
- FFTPACK – another Fortran FFT library (public domain)
- Architecture-specific:
  - Arm Performance Libraries[53]
  - Intel Integrated Performance Primitives
  - Intel Math Kernel Library
- Many more implementations are available,[54] for CPUs and GPUs, such as PocketFFT for C++

Other links:

- Odlyzko–Schönhage algorithm applies the FFT to finite Dirichlet series
- Schönhage–Strassen algorithm – asymptotically fast multiplication algorithm for large integers
- Butterfly diagram – a diagram used to describe FFTs
- Spectral music (involves application of DFT analysis to musical composition)
- Spectrum analyzer – any of several devices that perform spectrum analysis, often via a DFT
- Time series
- Fast Walsh–Hadamard transform
- Generalized distributive law
- Least-squares spectral analysis
- Multidimensional transform
- Multidimensional discrete convolution
- Fast Fourier Transform Telescope

References

1. Heideman, Michael T.; Johnson, Don H.; Burrus, Charles Sidney (1984). "Gauss and the history of the fast Fourier transform" (http://www.cis.rit.edu/class/simg716/Gauss_History_FFT.pdf) (PDF). *IEEE ASSP Magazine*. **1** (4): 14–21. CiteSeerX 10.1.1.309.181 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.309.181). doi:10.1109/MASSP.1984.1162257 (https://doi.org/10.1109%2FMASSP.1984.1162257). S2CID 10032502 (https://api.semanticscholar.org/CorpusID:10032502). Archived (https://web.archive.org/web/20130319053449/http://www.cis.rit.edu/class/simg716/Gauss_History_FFT.pdf) (PDF) from the original on 2013-03-19.
2. Van Loan, Charles (1992). *Computational Frameworks for the Fast Fourier Transform*. SIAM.
3. Strang, Gilbert (May–June 1994). "Wavelets". *American Scientist*. **82** (3): 250–255. JSTOR 29775194 (https://www.jstor.org/stable/29775194).
4. Kent, Ray D.; Read, Charles (2002). *Acoustic Analysis of Speech*. Singular/Thomson Learning. ISBN 0-7693-0112-6.
5. Dongarra, Jack; Sullivan, Francis (January 2000). "Guest Editors Introduction to the top 10 algorithms". *Computing in Science & Engineering*. **2** (1): 22–23. Bibcode:2000CSE.....2a..22D (https://ui.adsabs.harvard.edu/abs/2000CSE.....2a..22D). doi:10.1109/MCISE.2000.814652 (https://doi.org/10.1109%2FMCISE.2000.814652). ISSN 1521-9615 (https://www.worldcat.org/issn/1521-9615).

6. Gauss, Carl Friedrich (1866). "Theoria interpolationis methodo nova tractata" (https://babel.hathitrust.org/cgi/pt?id=uc1.c2857678;view=1up;seq=279) [Theory regarding a new method of interpolation]. *Nachlass* (Unpublished manuscript). Werke (in Latin and German). Vol. 3. Göttingen, Germany: Königlichen Gesellschaft der Wissenschaften zu Göttingen. pp. 265–303.
7. Heideman, Michael T.; Johnson, Don H.; Burrus, Charles Sidney (1985-09-01). "Gauss and the history of the fast Fourier transform". *Archive for History of Exact Sciences*. **34** (3): 265–277. CiteSeerX 10.1.1.309.181 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.309.181). doi:10.1007/BF00348431 (https://doi.org/10.1007%2FBF00348431). ISSN 0003-9519 (https://www.worldcat.org/issn/0003-9519). S2CID 122847826 (https://api.semanticscholar.org/CorpusID:122847826).
8. Yates, Frank (1937). "The design and analysis of factorial experiments". *Technical Communication No. 35 of the Commonwealth Bureau of Soils*. **142** (3585): 90–92. Bibcode:1938Natur.142...90F (https://ui.adsabs.harvard.edu/abs/1938Natur.142...90F). doi:10.1038/142090a0 (https://doi.org/10.1038%2F142090a0). S2CID 23501205 (https://api.semanticscholar.org/CorpusID:23501205).
9. Danielson, Gordon C.; Lanczos, Cornelius (1942). "Some improvements in practical Fourier analysis and their application to x-ray scattering from liquids". *Journal of the Franklin Institute*. **233** (4): 365–380. doi:10.1016/S0016-0032(42)90767-1 (https://doi.org/10.1016%2FS0016-0032%2842%2990767-1).
10. Lanczos, Cornelius (1956). *Applied Analysis* (https://archive.org/details/appliedanalysis00lanc_0). Prentice–Hall.
11. Cooley, James W.; Lewis, Peter A. W.; Welch, Peter D. (June 1967). "Historical notes on the fast Fourier transform". *IEEE Transactions on Audio and Electroacoustics*. **15** (2): 76–79. CiteSeerX 10.1.1.467.7209 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.467.7209). doi:10.1109/TAU.1967.1161903 (https://doi.org/10.1109%2FTAU.1967.1161903). ISSN 0018-9278 (https://www.worldcat.org/issn/0018-9278).
12. Cooley, James W.; Tukey, John W. (1965). "An algorithm for the machine calculation of complex Fourier series" (https://www.ams.org/mcom/1965-19-090/S0025-5718-1965-0178586-1/). *Mathematics of Computation*. **19** (90): 297–301. doi:10.1090/S0025-5718-1965-0178586-1 (https://doi.org/10.1090%2FS0025-5718-1965-0178586-1). ISSN 0025-5718 (https://www.worldcat.org/issn/0025-5718).
13. Cooley, James W. (1987). *The Re-Discovery of the Fast Fourier Transform Algorithm* (https://carma.newcastle.edu.au/jon/Preprints/Talks/CARMA-CE/FFT.pdf) (PDF). pp. 33–45. Archived (https://web.archive.org/web/20160820070623/https://carma.newcastle.edu.au/jon/Preprints/Talks/CARMA-CE/FFT.pdf) (PDF) from the original on 2016-08-20. {{cite book}}: |work= ignored (help)
14. Garwin, Richard (June 1969). "The Fast Fourier Transform As an Example of the Difficulty in Gaining Wide Use for a New Technique" (https://fas.org/rlg/690600-fft.pdf) (PDF). *IEEE Transactions on Audio and Electroacoustics*. AU-17 (2): 68–72. Archived (https://web.archive.org/web/20060517021147/http://www.fas.org/RLG/690600-fft.pdf) (PDF) from the original on 2006-05-17.
15. Rockmore, Daniel N. (January 2000). "The FFT: an algorithm the whole family can use". *Computing in Science & Engineering*. **2** (1): 60–64. Bibcode:2000CSE.....2a..60R (https://ui.adsabs.harvard.edu/abs/2000CSE.....2a..60R). CiteSeerX 10.1.1.17.228 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.17.228). doi:10.1109/5992.814659 (https://doi.org/10.1109%2F5992.814659). ISSN 1521-9615 (https://www.worldcat.org/issn/1521-9615). S2CID 14978667 (https://api.semanticscholar.org/CorpusID:14978667).

16. Frigo, Matteo; Johnson, Steven G. (January 2007) [2006-12-19]. "A Modified Split-Radix FFT With Fewer Arithmetic Operations". *IEEE Transactions on Signal Processing*. **55** (1): 111–119. Bibcode:2007ITSP...55..111J (https://ui.adsabs.harvard.edu/abs/2007ITSP...55..111J). CiteSeerX 10.1.1.582.5497 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.582.5497). doi:10.1109/tsp.2006.882087 (https://doi.org/10.1109%2Ftsp.2006.882087). S2CID 14772428 (https://api.semanticscholar.org/CorpusID:14772428).
17. Frigo, Matteo; Johnson, Steven G. (2005). "The Design and Implementation of FFTW3" (http://fftw.org/fftw-paper-ieee.pdf) (PDF). *Proceedings of the IEEE*. **93** (2): 216–231. Bibcode:2005IEEEP..93..216F (https://ui.adsabs.harvard.edu/abs/2005IEEEP..93..216F). CiteSeerX 10.1.1.66.3097 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.66.3097). doi:10.1109/jproc.2004.840301 (https://doi.org/10.1109%2Fjproc.2004.840301). S2CID 6644892 (https://api.semanticscholar.org/CorpusID:6644892). Archived (https://web.archive.org/web/20050207233032/http://www.fftw.org/fftw-paper-ieee.pdf) (PDF) from the original on 2005-02-07.
18. Gentleman, W. Morven; Sande, G. (1966). "Fast Fourier transforms—for fun and profit" (https://doi.org/10.1145%2F1464291.1464352). *Proceedings of the AFIPS*. **29**: 563–578. doi:10.1145/1464291.1464352 (https://doi.org/10.1145%2F1464291.1464352). S2CID 207170956 (https://api.semanticscholar.org/CorpusID:207170956).
19. Gauss, Carl Friedrich (1866) [1805]. *Theoria interpolationis methodo nova tractata* (https://gdz.sub.uni-goettingen.de/id/PPN235999628). Werke (in Latin and German). Vol. 3. Göttingen, Germany: Königliche Gesellschaft der Wissenschaften. pp. 265–327.
20. Brenner, Norman M.; Rader, Charles M. (1976). "A New Principle for Fast Fourier Transformation". *IEEE Transactions on Acoustics, Speech, and Signal Processing*. **24** (3): 264–266. doi:10.1109/TASSP.1976.1162805 (https://doi.org/10.1109%2FTASSP.1976.1162805).
21. Winograd, Shmuel (1978). "On computing the discrete Fourier transform" (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC430186). *Mathematics of Computation*. **32** (141): 175–199. doi:10.1090/S0025-5718-1978-0468306-4 (https://doi.org/10.1090%2FS0025-5718-1978-0468306-4). JSTOR 2006266 (https://www.jstor.org/stable/2006266). PMC 430186 (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC430186). PMID 16592303 (https://pubmed.ncbi.nlm.nih.gov/16592303).
22. Winograd, Shmuel (1979). "On the multiplicative complexity of the discrete Fourier transform". *Advances in Mathematics*. **32** (2): 83–117. doi:10.1016/0001-8708(79)90037-9 (https://doi.org/10.1016%2F0001-8708%2879%2990037-9).
23. Sorensen, Henrik V.; Jones, Douglas L.; Heideman, Michael T.; Burrus, Charles Sidney (1987). "Real-valued fast Fourier transform algorithms". *IEEE Transactions on Acoustics, Speech, and Signal Processing*. **35** (6): 849–863. CiteSeerX 10.1.1.205.4523 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.205.4523). doi:10.1109/TASSP.1987.1165220 (https://doi.org/10.1109%2FTASSP.1987.1165220).
24. Sorensen, Henrik V.; Jones, Douglas L.; Heideman, Michael T.; Burrus, Charles Sidney (1987). "Corrections to "Real-valued fast Fourier transform algorithms" ". *IEEE Transactions on Acoustics, Speech, and Signal Processing*. **35** (9): 1353. doi:10.1109/TASSP.1987.1165284 (https://doi.org/10.1109%2FTASSP.1987.1165284).
25. Heideman, Michael T.; Burrus, Charles Sidney (1986). "On the number of multiplications necessary to compute a length-2^n DFT". *IEEE Transactions on Acoustics, Speech, and Signal Processing*. **34** (1): 91–95. doi:10.1109/TASSP.1986.1164785 (https://doi.org/10.1109%2FTASSP.1986.1164785).
26. Duhamel, Pierre (1990). "Algorithms meeting the lower bounds on the multiplicative complexity of length-2^n DFTs and their connection with practical algorithms". *IEEE Transactions on Acoustics, Speech, and Signal Processing*. **38** (9): 1504–1511. doi:10.1109/29.60070 (https://doi.org/10.1109%2F29.60070).

27. Morgenstern, Jacques (1973). "Note on a lower bound of the linear complexity of the fast Fourier transform". *Journal of the ACM*. **20** (2): 305–306. doi:10.1145/321752.321761 (https://doi.org/10.1145%2F321752.321761). S2CID 2790142 (https://api.semanticscholar.org/CorpusID:2790142).
28. Pan, Victor Ya. (1986-01-02). "The trade-off between the additive complexity and the asynchronicity of linear and bilinear algorithms" (https://dl.acm.org/citation.cfm?id=8013). *Information Processing Letters*. **22** (1): 11–14. doi:10.1016/0020-0190(86)90035-9 (https://doi.org/10.1016%2F0020-0190%2886%2990035-9). Retrieved 2017-10-31.
29. Papadimitriou, Christos H. (1979). "Optimality of the fast Fourier transform" (https://doi.org/10.1145%2F322108.322118). *Journal of the ACM*. **26**: 95–102. doi:10.1145/322108.322118 (https://doi.org/10.1145%2F322108.322118). S2CID 850634 (https://api.semanticscholar.org/CorpusID:850634).
30. Lundy, Thomas J.; Van Buskirk, James (2007). "A new matrix approach to real FFTs and convolutions of length 2^k". *Computing*. **80** (1): 23–45. doi:10.1007/s00607-007-0222-6 (https://doi.org/10.1007%2Fs00607-007-0222-6). S2CID 27296044 (https://api.semanticscholar.org/CorpusID:27296044).
31. Haynal, Steve; Haynal, Heidi (2011). "Generating and Searching Families of FFT Algorithms" (https://web.archive.org/web/20120426031804/http://jsat.ewi.tudelft.nl/content/volume7/JSAT7_13_Haynal.pdf) (PDF). *Journal on Satisfiability, Boolean Modeling and Computation*. **7** (4): 145–187. arXiv:1103.5740 (https://arxiv.org/abs/1103.5740). Bibcode:2011arXiv1103.5740H (https://ui.adsabs.harvard.edu/abs/2011arXiv1103.5740H). doi:10.3233/SAT190084 (https://doi.org/10.3233%2FSAT190084). S2CID 173109 (https://api.semanticscholar.org/CorpusID:173109). Archived from the original (http://jsat.ewi.tudelft.nl/content/volume7/JSAT7_13_Haynal.pdf) (PDF) on 2012-04-26.
32. Duhamel, Pierre; Vetterli, Martin (1990). "Fast Fourier transforms: a tutorial review and a state of the art" (http://infoscience.epfl.ch/record/59946). *Signal Processing*. **19** (4): 259–299. doi:10.1016/0165-1684(90)90158-U (https://doi.org/10.1016%2F0165-1684%2890%2990158-U).
33. Edelman, Alan; McCorquodale, Peter; Toledo, Sivan (1999). "The Future Fast Fourier Transform?" (http://www.cs.tau.ac.il/~stoledo/Bib/Pubs/pp97-fft.pdf) (PDF). *SIAM Journal on Scientific Computing*. **20** (3): 1094–1114. CiteSeerX 10.1.1.54.9339 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.54.9339). doi:10.1137/S1064827597316266 (https://doi.org/10.1137%2FS1064827597316266). Archived (https://web.archive.org/web/20170705153832/http://www.cs.tau.ac.il/~stoledo/Bib/Pubs/pp97-fft.pdf) (PDF) from the original on 2017-07-05.
34. Guo, Haitao; Burrus, Charles Sidney (1996). Unser, Michael A.; Aldroubi, Akram; Laine, Andrew F. (eds.). "Fast approximate Fourier transform via wavelets transform". *Proceedings of SPIE*. Wavelet Applications in Signal and Image Processing IV. **2825**: 250–259. Bibcode:1996SPIE.2825..250G (https://ui.adsabs.harvard.edu/abs/1996SPIE.2825..250G). CiteSeerX 10.1.1.54.3984 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.54.3984). doi:10.1117/12.255236 (https://doi.org/10.1117%2F12.255236). S2CID 120514955 (https://api.semanticscholar.org/CorpusID:120514955).
35. Shentov, Ognjan V.; Mitra, Sanjit K.; Heute, Ulrich; Hossen, Abdul N. (1995). "Subband DFT. I. Definition, interpretations and extensions". *Signal Processing*. **41** (3): 261–277. doi:10.1016/0165-1684(94)00103-7 (https://doi.org/10.1016%2F0165-1684%2894%2900103-7).
36. Hassanieh, Haitham; Indyk, Piotr; Katabi, Dina; Price, Eric (January 2012). "Simple and Practical Algorithm for Sparse Fourier Transform" (https://www.mit.edu/~ecprice/papers/sparse-fft-soda.pdf) (PDF). *ACM-SIAM Symposium on Discrete Algorithms*. Archived (https://web.archive.org/web/20120304163958/http://www.mit.edu/~ecprice/papers/sparse-fft-soda.pdf) (PDF) from the original on 2012-03-04. (NB. See also the sFFT Web Page (http://groups.csail.mit.edu/netmit/sFFT/).)

37. Schatzman, James C. (1996). "Accuracy of the discrete Fourier transform and the fast Fourier transform" (http://portal.acm.org/citation.cfm?id=240432). *SIAM Journal on Scientific Computing*. **17** (5): 1150–1166. Bibcode:1996SJSC...17.1150S (https://ui.adsabs.harvard.edu/abs/1996SJSC...17.1150S). CiteSeerX 10.1.1.495.9184 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.495.9184). doi:10.1137/s1064827593247023 (https://doi.org/10.1137%2Fs1064827593247023).
38. Welch, Peter D. (1969). "A fixed-point fast Fourier transform error analysis". *IEEE Transactions on Audio and Electroacoustics*. **17** (2): 151–157. doi:10.1109/TAU.1969.1162035 (https://doi.org/10.1109%2FTAU.1969.1162035).
39. Ergün, Funda (1995). "Testing multivariate linear functions". *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing - STOC '95*. Kyoto, Japan. pp. 407–416. doi:10.1145/225058.225167 (https://doi.org/10.1145%2F225058.225167). ISBN 978-0897917186. S2CID 15512806 (https://api.semanticscholar.org/CorpusID:15512806).
40. Nussbaumer, Henri J. (1977). "Digital filtering using polynomial transforms". *Electronics Letters*. **13** (13): 386–387. Bibcode:1977ElL....13..386N (https://ui.adsabs.harvard.edu/abs/1977ElL....13..386N). doi:10.1049/el:19770280 (https://doi.org/10.1049%2Fel%3A19770280).
41. Mohlenkamp, Martin J. (1999). "A Fast Transform for Spherical Harmonics" (http://www.ohiouniversityfaculty.com/mohlenka/research/MOHLEN1999P.pdf) (PDF). *Journal of Fourier Analysis and Applications*. **5** (2–3): 159–184. CiteSeerX 10.1.1.135.9830 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.135.9830). doi:10.1007/BF01261607 (https://doi.org/10.1007%2FBF01261607). S2CID 119482349 (https://api.semanticscholar.org/CorpusID:119482349). Archived (https://web.archive.org/web/20170506033135/http://ohiouniversityfaculty.com/mohlenka/research/MOHLEN1999P.pdf) (PDF) from the original on 2017-05-06. Retrieved 2018-01-11.
42. "libftsh library" (https://web.archive.org/web/20100623034953/http://www.math.ohiou.edu/~mjm/research/libftsh.html). Archived from the original (http://www.math.ohiou.edu/~mjm/research/libftsh.html) on 2010-06-23. Retrieved 2007-01-09.
43. Rokhlin, Vladimir; Tygert, Mark (2006). "Fast Algorithms for Spherical Harmonic Expansions" (http://tygert.com/sph2.pdf) (PDF). *SIAM Journal on Scientific Computing*. **27** (6): 1903–1928. Bibcode:2006SJSC...27.1903R (https://ui.adsabs.harvard.edu/abs/2006SJSC...27.1903R). CiteSeerX 10.1.1.125.7415 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.125.7415). doi:10.1137/050623073 (https://doi.org/10.1137%2F050623073). Archived (https://web.archive.org/web/20141217212000/http://tygert.com/sph2.pdf) (PDF) from the original on 2014-12-17. Retrieved 2014-09-18. [1] (http://www.cs.yale.edu/publications/techreports/tr1309.pdf)
44. Potts, Daniel; Steidl, Gabriele; Tasche, Manfred (2001). "Fast Fourier transforms for nonequispaced data: A tutorial" (http://www.tu-chemnitz.de/~potts/paper/ndft.pdf) (PDF). In Benedetto, J. J.; Ferreira, P. (eds.). *Modern Sampling Theory: Mathematics and Applications*. Birkhäuser. Archived (https://web.archive.org/web/20070926222858/http://www.tu-chemnitz.de/~potts/paper/ndft.pdf) (PDF) from the original on 2007-09-26.
45. Burgess, Richard James (2014). *The History of Music Production* (https://books.google.com/books?id=qMKiAwAAQBAJ). Oxford University Press. ISBN 978-0199357178. Retrieved 1 August 2019.
46. Chu, Eleanor; George, Alan (1999-11-11) [1999-11-11]. "Chapter 16". *Inside the FFT Black Box: Serial and Parallel Fast Fourier Transform Algorithms*. CRC Press. pp. 153–168. ISBN 978-1-42004996-1.
47. Fernandez-de-Cossio Diaz, Jorge; Fernandez-de-Cossio, Jorge (2012-08-08). "Computation of Isotopic Peak Center-Mass Distribution by Fourier Transform". *Analytical Chemistry*. **84** (16): 7052–7056. doi:10.1021/ac301296a (https://doi.org/10.1021%2Fac301296a). ISSN 0003-2700 (https://www.worldcat.org/issn/0003-2700). PMID 22873736 (https://pubmed.ncbi.nlm.nih.gov/22873736).

48. Minenna, Marcello (October 2008). "A revisited and stable Fourier transform method for affine jump diffusion models". *Journal of Banking and Finance*. **32** (10): 2064–2075. doi:10.1016/j.jbankfin.2007.05.019 (https://doi.org/10.1016%2Fj.jbankfin.2007.05.019).
49. Cormen, Thomas H.; Nicol, David M. (1998). "Performing out-of-core FFTs on parallel disk systems". *Parallel Computing*. **24** (1): 5–20. CiteSeerX 10.1.1.44.8212 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.44.8212). doi:10.1016/S0167-8191(97)00114-2 (https://doi.org/10.1016%2FS0167-8191%2897%2900114-2). S2CID 14996854 (https://api.semanticscholar.org/CorpusID:14996854).
50. Dutt, Alok; Rokhlin, Vladimir (1993-11-01). "Fast Fourier Transforms for Nonequispaced Data". *SIAM Journal on Scientific Computing*. **14** (6): 1368–1393. Bibcode:1993SJSC...14.1368D (https://ui.adsabs.harvard.edu/abs/1993SJSC...14.1368D). doi:10.1137/0914081 (https://doi.org/10.1137%2F0914081). ISSN 1064-8275 (https://www.worldcat.org/issn/1064-8275).
51. Rockmore, Daniel N. (2004). "Recent Progress and Applications in Group FFTs". In Byrnes, Jim (ed.). *Computational Noncommutative Algebra and Applications*. NATO Science Series II: Mathematics, Physics and Chemistry. Vol. 136. Springer Netherlands. pp. 227–254. CiteSeerX 10.1.1.324.4700 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.324.4700). doi:10.1007/1-4020-2307-3_9 (https://doi.org/10.1007%2F1-4020-2307-3_9). ISBN 978-1-4020-1982-1. S2CID 1412268 (https://api.semanticscholar.org/CorpusID:1412268).
52. Ryo, Asaka; Kazumitsu, Sakai; Ryoko, Yahagi (2020). "Quantum circuit for the fast Fourier transform" (https://link.springer.com/article/10.1007/s11128-020-02776-5). *Quantum Information Processing*. **19** (277): 277. arXiv:1911.03055 (https://arxiv.org/abs/1911.03055). Bibcode:2020QuIP...19..277A (https://ui.adsabs.harvard.edu/abs/2020QuIP...19..277A). doi:10.1007/s11128-020-02776-5 (https://doi.org/10.1007%2Fs11128-020-02776-5). S2CID 207847474 (https://api.semanticscholar.org/CorpusID:207847474).
53. "Arm Performance Libraries" (https://www.arm.com/products/development-tools/server-and-hpc/allinea-studio/performance-libraries). Arm. 2020. Retrieved 2020-12-16.
54. "Complete list of C/C++ FFT libraries" (https://community.vcvrack.com/t/complete-list-of-c-c-fft-libraries/9153). *VCV Community*. 2020-04-05. Retrieved 2021-03-03.

Further reading

- Brigham, E. Oran (2002). *The Fast Fourier Transform*. New York: Prentice-Hall.
- Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2001). "Chapter 30: Polynomials and the FFT". *Introduction to Algorithms* (2 ed.). MIT Press / McGraw-Hill. ISBN 0-262-03293-7.
- Elliott, Douglas F.; Rao, K. Ramamohan (1982). *Fast transforms: Algorithms, analyses, applications*. New York, USA: Academic Press.
- Guo, Haitao; Sitton, Gary A.; Burrus, Charles Sidney (1994). "The quick discrete Fourier transform". *Proceedings of ICASSP '94. IEEE International Conference on Acoustics, Speech and Signal Processing*. Vol. 3. pp. 445–448. doi:10.1109/ICASSP.1994.389994 (https://doi.org/10.1109%2FICASSP.1994.389994). ISBN 978-0-7803-1775-8. S2CID 42639206 (https://api.semanticscholar.org/CorpusID:42639206).
- Johnson, Steven G.; Frigo, Matteo (2007). "A modified split-radix FFT with fewer arithmetic operations" (http://www.fftw.org/newsplit.pdf) (PDF). *IEEE Transactions on Signal Processing*. **55** (1): 111–119. Bibcode:2007ITSP...55..111J (https://ui.adsabs.harvard.edu/abs/2007ITSP...55..111J). CiteSeerX 10.1.1.582.5497 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.582.5497). doi:10.1109/tsp.2006.882087 (https://doi.org/10.1109%2Ftsp.2006.882087). S2CID 14772428 (https://api.semanticscholar.org/CorpusID:14772428). Archived (https://web.archive.org/web/20050526050219/http://www.fftw.org/newsplit.pdf) (PDF) from the original on 2005-05-26.

- Press, William H.; Teukolsky, Saul A.; Vetterling, William T.; Flannery, Brian P. (2007). "Chapter 12. Fast Fourier Transform" (http://apps.nrbook.com/empanel/index.html#pg=600). *Numerical Recipes: The Art of Scientific Computing* (3 ed.). New York, USA: Cambridge University Press. ISBN 978-0-521-88068-8.
- Singleton, Richard Collom (June 1969). "A Short Bibliography on the Fast Fourier Transform". *Special Issue on Fast Fourier Transform*. Vol. AU-17. IEEE Audio and Electroacoustics Group. pp. 166–169. doi:10.1109/TAU.1969.1162029 (https://doi.org/10.1109%2FTAU.1969.1162029). (NB. Contains extensive bibliography.)
- Elena Prestini: "The Evolution of Applied Harmonic Analysis", Springer, ISBN 978-0-8176-4125-2 (2004), Sec.3.10 'Gauss and the asteroids: history of the FFT'.

External links

- Fast Fourier Transform for Polynomial Multiplication (http://www.cs.pitt.edu/~kirk/cs1501/animations/FFT.html) – fast Fourier algorithm
- *Fast Fourier Transforms (http://cnx.org/content/col10550/)*, Connexions online book edited by Charles Sidney Burrus, with chapters by Charles Sidney Burrus, Ivan Selesnick, Markus Pueschel, Matteo Frigo, and Steven G. Johnson (2008)
- Fast Fourier transform — FFT (http://www.librow.com/articles/article-10) – FFT programming in C++ – the Cooley–Tukey algorithm
- Online documentation, links, book, and code (https://www.jjj.de/fxt/)
- Sri Welaratna, "Thirty years of FFT analyzers (http://www.dataphysics.com/30_Years_of_FFT_Analyzers_by_Sri_Welaratna.pdf) Archived (https://web.archive.org/web/20140112235745/http://www.dataphysics.com/30_Years_of_FFT_Analyzers_by_Sri_Welaratna.pdf) 2014-01-12 at the Wayback Machine", *Sound and Vibration* (January 1997, 30th anniversary issue) – a historical review of hardware FFT devices
- ALGLIB FFT Code (http://www.alglib.net/fasttransforms/fft.php) – a dual/GPL-licensed multilanguage (VBA, C++, Pascal, etc.) numerical analysis and data processing library
- SFFT: Sparse Fast Fourier Transform (http://groups.csail.mit.edu/netmit/sFFT/) – MIT's sparse (sub-linear time) FFT algorithm, sFFT, and implementation
- VB6 FFT (https://web.archive.org/web/20130928020959/http://www.borgdesign.ro/fft.zip) – a VB6 optimized library implementation with source code
- Interactive FFT Tutorial (https://www.karlsims.com/fft.html) – a visual interactive intro to Fourier transforms and FFT methods

---